A Multi-Agent Approach for Complex System Design

Irène Degirmenciyan-Cartault
Dassault Aviation, 78, quai Marcel Dassault Cedex 300 92552 St Cloud Cedex France
irene.degirmenciyan@dassault-aviation.fr

Abstract: Multi-agent systems that arose from research in Distributed Artificial Intelligence are now considered as a new paradigm to design and model complex systems. The design of complex systems seems to be more intuitive with cognitive agents because the designer works at a high level of abstraction where the agent is an important granularity entity with aspects of calculation, reasoning and control which make it quasi autonomous. After a brief overview of the required notions to understand the multi-agent systems' domain, we describe the JACK agent-oriented environment on top of which we are developing the SCALA environment that provides a multiagent-based methodology and tool for the design of complex systems. In SCALA, the global behaviour (*i.e.* its goal) of the system is modelled through a functional approach based on the definition of a graph of dependencies between the basic behaviours (*i.e.* tasks to accomplish to achieve a goal). The definition of this graph provides the necessary knowledge to manage cooperation between the agents and to plan reactively their activities when new events occur and when they have to reorganise themselves.

1. Introduction

Multi-agent systems that arose from research in Distributed Artificial Intelligence are now considered as a new paradigm to design and model complex systems. Those systems are used for several well-known reasons: their ability to run in unpredictable or less predictable environments, to react to unknown events, to reason, to cooperate, to learn and to be pro-active. The design of complex systems seems to be more intuitive with cognitive agents because the designer works at a high level of abstraction where the agent is an important granularity entity with aspects of calculation, reasoning and control which make it quasi autonomous. Then the distribution of the control on the agents gives the system more robustness, efficiency and a better reactivity thanks to sophisticated coordination mechanisms. Agent-oriented programming addresses the need for software systems to exhibit rational, human-like behaviour in their respective problem domains. Traditional software systems make it difficult to model rational behaviour, and often programs written in these systems experience limitations, especially when attempting to operate in real time environment.

The aim of this paper is not to present the entire domain of research on multi-agent systems, but to point out characteristics that seem to be relevant for the design of complex systems. So, in section 2 we overview the required major notions to understand the multi-agent systems' domain. In section 3 we describe an example of the JACK agent-oriented environment on top of which we develop the SCALA project that provides a multiagent-based methodology and a tool for the design of complex system (section 4). Then, we expose in section 5 the perspectives of this work to extend it to time constraints.

2. Multi-agent Systems

In this part, we introduce the notion of **agent** and the advantages to adopt such a technology of programming.

2.1. Definitions

Although there is no consensus on the agent definition, we can assume that an agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its designed objective [WOO 95].

Paper presented at the RTO AVT Course on “Intelligent Systems for Aeronautics”, held in Rhode-Saint-Genèse, Belgium, 13-17 May 2002, and published in RTO-EN-022.

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **01 JUN 2003**	2. REPORT TYPE **N/A**	3. DATES COVERED **-**
4. TITLE AND SUBTITLE **A Multi-Agent Approach for Complex System Design**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Dassault Aviation, 78, quai Marcel Dassault Cedex 300 92552 St Cloud Cedex France**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release, distribution unlimited**		
13. SUPPLEMENTARY NOTES **See also ADM001519. RTO-EN-022**		
14. ABSTRACT		
15. SUBJECT TERMS		

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**UU**	**14**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

The notion of autonomy means that agents have control both over their own internal state, and over their behaviour. They act without the intervention of humans or other systems.
To define the notion of **intelligent agent**, the notion of flexibility is added. Intelligent agents can be characterised by their ability to carry on flexible autonomous action [JEN 98], *i.e.* they present:

- Reactivity: intelligent agents are able to perceive their environment, and respond in a timely fashion to changes that occur in it to satisfy their design objectives;
- Pro-activeness: intelligent agents are able to exhibit goal-directed behaviour by taking the initiative in order to satisfy their design objectives;
- Social ability: are capable of interacting with other agents (and possibly humans) in order to satisfy their design objectives.

In a dynamic and uncertain environment, these characteristics are not easy to obtain. The agents have to continually respond to their environment to reactively take into account its changes, while maintaining their goals. The difficulty for an agent is to deal simultaneously with these two aspects. The agents thus have to attempt to achieve their goals, but not in a blindly fashion. They continually have to perceive the environment and be ready to quickly react to new situations. The carried on goal can be questioned, or some changes in the environment can impact the way to achieve it, etc.
The last important point is social ability. Agents have to negotiate or cooperate with others to achieve shared goal. This social ability includes the possibility to define organisational structure where agents hold roles and obey to social rules. The multi-agent approach specially focuses on these aspects.

A **multi-agent system** is a distributed system composed by several agents interacting with each other's. A multi-agent can be characterised by the following points as reminded in [BRIOT 01]:

- Each agent has limited information or capabilities;
- There is no global control of the multi-agent system;
- The data are decentralised;
- The calculation is asynchronous.

Multi-agent systems have the traditional advantages of the distributed and concurrent problem solving, such as modularity, computation time (parallelism) and reliability (due to the redundancy of resources and capabilities). Multi-agent systems are also interested in developing sophisticated interaction mechanisms, such as cooperation, coordination and negotiation.

2.2. Agents versus objects

When one considers only the relative properties of agents and objects, one can have difficulties to see the added value of agents. Indeed objects are computational entities that encapsulate some states, are able to perform actions (methods), and communicate with other objects through message passing. Yet there are significant differences between agents and objects [WOO 99]:

The principle of encapsulation is the idea that objects can have control over their own states, *i.e.* they exhibit a certain autonomy over their states. But an object does not exhibit control over its behaviour. A method *m* of an object is executed by this object when another object invokes it. The object that performs the action (method *m*) has no control over this invocation. The decision lies with the object that invokes the method. In the agent case, we do not think of invocation, but in requesting actions. The decision lies with the agent that receives the request. The agents have the control over the decision to execute an action or not.

Although one can integrate flexible (reactive, pro-active, social) autonomous behaviour in object-oriented programs, this kind of behaviour is not inherent to the object philosophy, and this means to implement agents' behaviour in an object-oriented language.

Another important distinction is that agents have each their own thread of control, whereas there is only one in an object-oriented program. Even if we take into account the concept of concurrency that arrives in object-oriented languages, such as the multi-threading in Java, it does not capture the idea of autonomous entities. A multi-agent system is inherently multi-threaded.

3. JACK: An agent infrastructure [HOW 01]

A host of environments support the development of multi-agent applications each presenting different characteristics. An effort of standardisation is led by the FIPA (Foundation for Intelligent and Physical Agent).

In this section, we are describing one of them, the agent-oriented language JACK software, on which lies our work presented in section 4. JACK is an agent-oriented development environment build on top of Java. It addresses the need for software systems to exhibit rational, human-like behaviour in their respective problem domains. The agents used in JACK are intelligent agents. They model reasoning behaviour according to the theoretical Belief Desire Intention (BDI) model of Artificial Intelligence developed by Rao and Georgeff [RAO 95]. More precisely, following the BDI model the JACK agents are autonomous entities that have goals to achieve thanks to events to which they are sensitive. To determine the way to achieve a goal, the agents have plans. Each plan describes the way the agent has to react to cope with a given situation. Consequently, an agent attempt to achieve a goal (its desire) using the relevant plan (its intention) that it will have determined in analysing its knowledge (its beliefs) on the external environment.

So, a JACK agent can exhibit reasoning behaviour under both pro-active (goal directed) and reactive (event driven) stimuli. Each agent has:

- A set of beliefs about the world;
- A set of events that it will respond to;
- A set of goals that it may arise to achieve;
- A set of plans that describe how it can handle the goals or events that may arise.

Let us now detail JACK specific classes [JAR 01]:

The Agent class

The Agent class embodies all the functionality associated with a JACK intelligent agent. It allows to define the behaviour of an agent, its capabilities, the type of messages and events to which it is sensitive and the plans it uses to achieve its goals. Each JACK agent is associated to an independent execution process (a Java thread).

In general, the definition of this class needs to include the following conceptual statements:

- *Knowledge Bases* which the agent can use and refer to
- *Events* (both internal and external) that the agent is prepared to handle
- *Plans* that the agent can execute
- *Events* the agent can post **internally** (to be handled by other plans)
- *Events* the agent can send **externally** to other agents.

The Database Class

The *Database* class implements the main data storage device that agents use. Each database class describes a set of *beliefs* that the agent can have. It represents these beliefs in a *first order*, *tuple-based relational* database system. The logical consistency of the belief this database contains is automatically maintained. Hence, for example, if an agent adds a belief that contradicts a belief it already has, the database class detects this and automatically ejects the old belief. The database class is not the only way that an agent can represent information. Agents can also include ordinary members and other data storage structures that have been implemented in Java.

The Event class

Events motivate an agent to take action. There are a number of event types in JACK, each with different uses. These different event types help model.

- Internal stimuli; essentially events that an agent sends to itself. These internal events are integral to the ongoing execution of an agent and the reasoning that it undertakes.
- External stimuli, such as messages from other agents, or percepts that an agents receives from its environment.
- *Motivations* that the agent may have, such as goals that the agent is committed to achieving.

Events are the origin of all activity within an agent-oriented system. Whenever an event occurs, an agent initiates a task to handle it. This task can be thought of as a thread of activity within the agent. This task causes the agent to choose between the plans it has available, executing a selected plan or plan set (depending on the event processing model chosen) until it succeeds or fails. If plan execution succeeds, then the event that initiated it is said to have succeeded. If plan execution fails, on the other hand, there are two options. Under normal event handling the event is said to have failed after the first instance of plan failure. Under *BDI* event handling, a number of plans can be selected for execution and these are attempted in turn, in order to try to achieve successful plan execution. If the set of chosen plans is exhausted, then the event is said to have failed. There are a number of event classes in the JACK Agent Language, each representing different types of motivation to act.

One of the important aspects of the BDI reasoning model at a conceptual level is that it models *goal-directed behaviour* in agents, rather than plan-directed behaviour. That is, an agent *commits* to the desired outcome, not the method chosen to achieve it. When using the BDI reasoning model, an agent does not simply react to incoming information, but sets itself a goal that it then tries to achieve. Rather than distracting an agent from its goal, incoming events are added to an agents knowledge base and can then subtly influence its behaviour.

The key difference between normal events and BDI events is how an agent selects plans for execution. With normal events, the agent selects the first applicable plan instance for a given event and executes that plan instance only. The handling of BDI events is more complex and powerful. An agent can assemble a plan set for a given event, apply sophisticated heuristics for plan choice and act intelligently on plan failure. At least one of the following characteristics applies to each type of BDI event under the BDI model:

- *Meta-level reasoning* : it allows precise control over how an agent chooses a plan for execution from the set of applicable plans. Whenever there is more than one applicable plan instance for a given BDI event, a *special* event is posted within the agent. By choosing to handle this event an agent can implement meta-level reasoning. If the meta-level reasoning plan fails and does not select a plan for execution, then the default plan selection method is invoked.
- *Reconsidering alternative plans on plan failure* : if a course of action (plan) fails, an agent can try a number of other courses of action by attempting any number of applicable plans to achieve the goal that has been set.
- *Re-calculating the applicable plan set* : after that the previous selected plan has failed, an agent may select an alternative plan: it either keeps track of the plan instances that were initially applicable and select another member of this set; or it re-computes which plan instances are applicable and select one from the new set, excluding plan instances that have already failed.

Additionally it is possible to further control BDI behaviour by setting *behaviour attributes*.

The Plan class

The Plan class describes a sequence of actions that an agent can take when an event occurs. Whenever an event is posted and the agent adopts a task to handle it, the first thing that the agent does is to try to find a plan to handle the event. Plans are similar to methods and functions from more conventional programming languages. Each plan is capable of handling a single event. An agent may further discriminate between plans that declare they handle an event by determining whether a plan is *relevant* by using specific JACK methods that can be assimilated to a filter.

The Capability Class

The *capability* concept is a means of structuring reasoning elements of agents into clusters that implement selected reasoning capabilities. This technique simplifies agent system design, allows code reuse and encapsulation of agent functionality. Capabilities represent functional aspects of an agent that can be plugged in as required. This *capability as component* approach allows an agent system architect to build up a library of capabilities over time. These components can then be used to add selected functionality to an agent. Events, databases, plans, Java code and other capabilities can all be combined to make a capability.

4. SCALA: A methodology and a tool to design complex systems

At Dassault Aviation, we work on a project named SCALA (Cooperative System of Software Autonomous Agent) developed on top of the JACK agent-oriented software presented in section 3. SCALA aims at showing the interest of the multi-agent approach for modelling and designing complex systems, where several entities have to cooperate to achieve joint goals. SCALA provides a tool and a methodology that enable the designer to build at a high-level of abstraction the behaviour of a multi-agent systems.

The global behaviour of the system (*i.e.* its goals) is modelled through a functional approach based on the definition of the graph of dependencies between basic behaviours, *i.e.* the tasks to be accomplished by the agents of the system.

4.1. The objectives

The major objectives carried out in the SCALA project are:

- To provide a tool to prototype multi-agent systems (MAS) in proposing a methodology of design based on the functional requirements;
- To make easier the modelling and the design of such systems thanks to a high level abstraction language;
- To simulate different types of organisation and communication protocols between agents;
- To constitute and provide libraries of reusable mechanisms and protocols;
- To propose tools to support the design of the system and to monitor the behaviour of the system (individual and collective behaviour).

In next section, we are focusing on the methodology proposed in SCALA to assist the design of complex systems.

4.2. A methodology to develop complex systems

This methodology is based on a functional approach enabling designers to elaborate and model the whole behaviour of the multi-agent system. This methodology is composed of 8 steps:

1. Definition of the graph of functional dependencies;
2. Definition of the sub-graphs (and the goals associated);
3. Identification of the expected relevant events;
4. Description of the elementary behaviours or tasks to be accomplished by the agents;
5. Definition of the agents;
6. Definition of the group of agents;
7. Choice of the social organisations;
8. Choice of the cooperation protocols.

Now, we are going to detail the different points of this methodology and briefly describe the functioning of SCALA through a given type of application: the simulation of air combat missions. This presentation allows us to point out several limitations that we are attempting to overcome in the second part of the presented study.

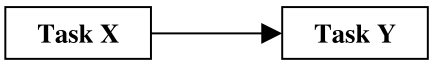
4.3. The graph of functional dependencies

The graph of dependencies enables the designer to model at a high level of abstraction the behaviour of the multi-agent system. This graph is constituted by one or more distinct graphs composed of tasks or basic behaviours that have to be accomplished by the agents of the system, and the constraints between them. The definition of the graph provides the necessary knowledge to manage the cooperation between the agents. Different types of the control either centralised or distributed, can be built depending on which agent(s) own(s) this knowledge. At this stage, tasks are not allocated yet to the agents. The assignment is made dynamically (see § 4.11) according to the current situation and the available resources (agents are assimilated to resources through their skills). Thus we give the agents a certain freedom of action that makes the system more reactive and allows to avoid some failures such as the use of non-available resources. We have a proscriptive approach that constrains the behaviours rather a directed one. For example, according to the constraints on a task, an agent can attempt to ask another agent for help or performs another task which does not requires the help of another agent .
The graph of dependencies is in fact, at a given level of abstraction, a decomposition of the global behaviour of the system, close to a decomposition of a problem into sub-tasks, and including the different possible alternatives for each sub-goal. In that way, the SCALA agents' behaviour is task-oriented.
Furthermore, we will see, in § 4.7 that each task is associated to several JACK plans, that specify the different methods to accomplish a same task. By default, *i.e.* without specific associated constraints, each task of the graph has to be achieved by a single agent.

Let us now detail the constraints between tasks. They are defined by links or connectors expressing notions of synchronism (at the beginning or at the end of several tasks), exclusion (the execution of one task inhibits others'), refinement (several methods can be invoked to execute a task) and abstraction (a task is composed of others). Another type of constraint is about the number of agents required to perform a same task.

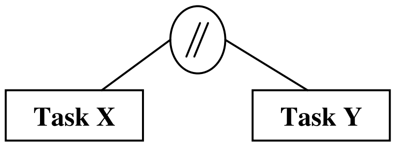
The link of precedence:

This link shows that Task Y cannot be executed before Task X. A task may have several links of precedence and may also be the predecessor of several tasks.

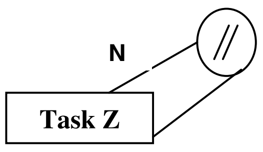


The connector of temporal synchronisation //:

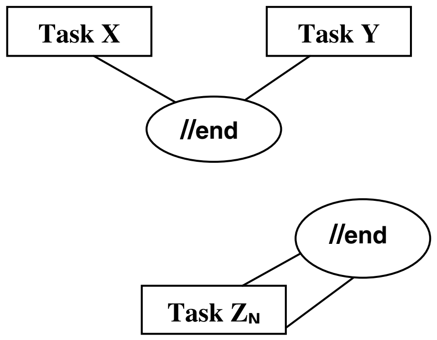
This connector implies that Task X and Task Y begin synchronously.



It is also possible to define that N agents have to execute the same task concurrently. It is the case for the following example where N agents have to begin Task Z synchronously.



The connector of temporal synchronisation //end:

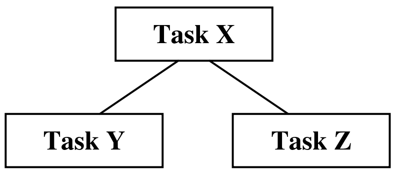


This connector implies a synchronisation of termination. Task X and Y must end at the same instant, or N agents have to end their task at the same instant.

Remark: To realise this connector, the graph must bring the notion of delays on tasks. But it could be applied easily in case of interruptible tasks.

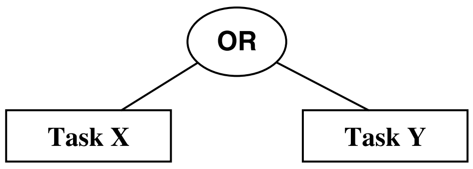
The link of decomposition:

The decomposition may be considered as a refinement.



Task X (abstracted Task) is composed of two sub-tasks Y and Z. The execution of Y and Z is necessary to validate Task X. The execution of Y and Z is possible only when the predecessors of X have been realised.

The connector OR:

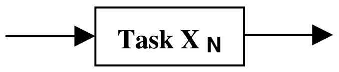


This connector represents the concept of exclusion between tasks. The execution of one of the tasks will prevent the execution of the other linked tasks.

In this example, the activation of Task X will inhibit Task Y one. The choice of the branch is context dependant.

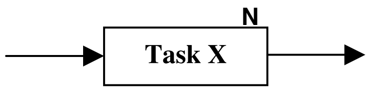
The connector T_N:

This connector implies that the task must be executed N times but with no constraints on the executors. So a single, or many agents may execute N times Task X.

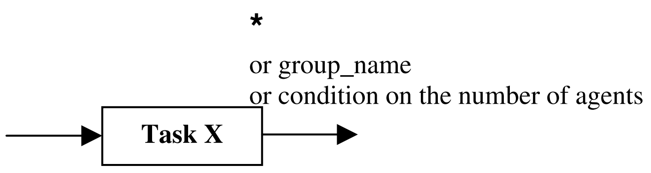


Constraints of allocation:

This connector means Task X must be done by N agents but with no constraints of synchronisation.

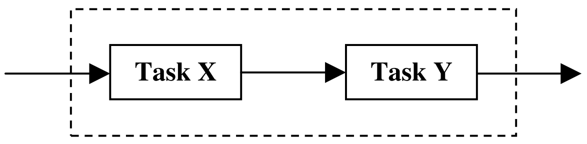


When all the agents of the group have to execute this task, the N is replaced by a star (*). This permits to leave the constraint on the number of agents.
But one can also precise if a group has to do this task: *group_name* or you can put conditions on the number of agent that has to do the task: *≥2 (this condition means that at least two agents of the group have to do the task).



Series:

A series of tasks must be executed by the same agent or the same group of agents.
In this example, task X and Y must be executed by a same agent.



Those all connectors can be merged to express more sophisticated constraints between tasks.

4.4. The definition of the sub-graphs (goals)

The sub-graphs represent partial graphs of the graph of functional dependencies. Each sub-graph is associated to a goal that replies to a particular event. For example, "**To intercept a threat**" is one of the possible goals to be achieved by the patrol if the event "**New contact on the radar**" occurs. This goal is associated to a sub-graph (a set of tasks and connectors) depicting the functional possible procedures the agents have to carry out to achieve it.

4.5. The expected relevant events

The agents of the system have sometimes to reorganise themselves to achieve goals. The goals are generally fulfilled by one agent or a group of agents. The agents take into account new events that can be interactively provided by the designer during the simulation (for example the on-line creation of a new threat) coming either from environment changes, or from messages between agents. A new event implies that the agents have to cope with a new sub-graph. All the difficulty lies in the management of the new goals. Each event is characterised by a degree of priority which allows the agent to select the most urgent to treat. The algorithm concerned is developed in fig. 6. To model the system, the designer specifies the different events to which the agents are sensitive. The designer has to specify the "event/goal" pairs.

Now, let us present our scenario to illustrate the methodology of SCALA and model the behaviour of the designed multi-agent system.

4.6. An interception scenario

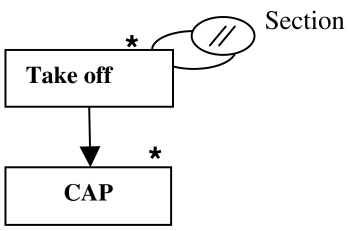


fig. 1: Sub-graph "CAP"

In the scenario, four aircraft fly as a division on a close-air support mission. A division is composed of two sections, each containing two aircraft. Close-air support missions involve substantial communication and coordination to ensure the success of the mission. We are interested to a high level of coordination, *i.e.* on the goal to achieve and actions to perform. Command and control are provided by an airborne radar plane (in the friend area) and a number of forward air controllers (AC), friend forces on the enemy ground. We are following this scenario over the steps of SCALA's methodology.

The division's pre-briefed goal is **"Bomb Target 1"**. The aircraft take off from the carrier and rendezvous at a pre-briefed location to join into formation. The rendezvous is necessary to coordinate group flight because we assume that only two planes (a section) can launch simultaneously. According to the formalism defined in SCALA, this can be modelled by the sub-graph (fig.1). The star (*) means that all the sections of the group defined by the designer have to take off. This implies no conditions on the number of aircraft involved in the mission (it is specified via the agents definition) but only the (functional) way they have to take off. This generic approach of specifying behaviour

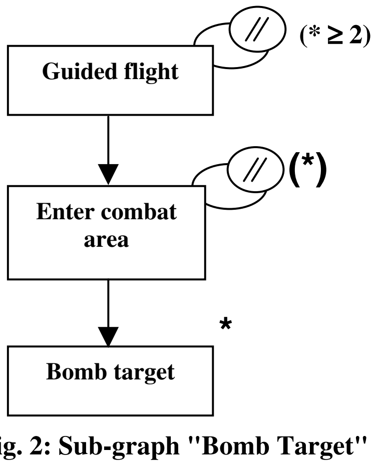


fig. 2: Sub-graph "Bomb Target"

ensure the re-usability of the graphs. The second task "**CAP**" (Control Air Patrol) expresses, in the same way, that all the planes have to join at a pre-briefed point.

While flying their route, the lead of the division checks in with the air controllers any changes in routing and the permission to enter the combat area. Once the AC verifies the mission, visually acquires the planes, and determines they can bomb the target without endangering friendly forces, the AC gives the final permission to drop their ordnance. Then, they exit the area and fly back on their egress route. The constraints (//) and (*** ≥ 2**) express that the task "**Guided flight**" has to be accomplished simultaneously by a group composed of at least two aircraft. This task is monitored by the airborne radar plane. All the agents of the related group must synchronously begin the task "Enter combat area". Then, all the agents, who carry a bomb, have to drop it (not necessarily at the same time). We do not have to precise on this graph that the last task is to return to the base because it is the task by default: it is a "home state" automatically executed. The global plan is the concatenation of these two sub-graphs (fig. 1 & fig. 2). In this step of the modelling, the designer has to define the "event/goal" pairs: "**Order to bomb the target / Bomb the target**". We further expose a complete scenario with new events occurring during the simulation.

The next step is to define the local behaviours, *i.e.* the different tasks.

4.7. Definition of the tasks

The tasks have some notable properties that we are describing here.

A set of methods

Each task is associated to a set of methods. A task is in fact a sub-goal that can be achieved by different manners. For instance, several tactics lead to the same result, but their use is context-dependent and an agent have to decide which of them apply according to its skills or to the availability of other agents in the case of a team-tactics. (*Remark*: this functionality is directly available in JACK with the meta reasoning of the agents based on sophisticated heuristics for the selection of the relevant plans).

Interruptible tasks

The tasks can be interruptible or not interruptible. The interruptible tasks have the capability to stop their execution and then to resume it or to definitely abandon it.
For example, a task can be interrupted when an event implies the interruption of the current method (same remark than below). In the related scenario, the "**Guided Flight**" task is interrupted to let the agents enter in the combat area when they arrive close to this zone and when they had received the permission from the AC. Furthermore, the execution of a task is submitted to constraints clustered in pre and post conditions.

Pre and Post conditions

Those notions are linked to the perception of the agent or its knowledge. The pre-conditions specify the conditions to execute a task but also the way. In fact, several manners can exist to achieve a task. Those pre-conditions can be assimilated to a filter for the choice of the relevant method.
The post-conditions describe the changes of the resources after the execution of the task. For example, if a missile is launched, the number of missiles must decrease in the knowledge base of the concerned agent.
According to our scenario, the pre-conditions on the task "**Enter combat area**" are to get the permission from the AC. For the task "**Bomb the target**", the planes have to wait for the permission and to be sure of the target to bomb.

Number of agents

It is the necessary number of agents required to execute the task. This information is very important to initiate cooperation between agents.

Skills and level of specialisation

Here are defined the necessary skills and level of specialisation required to execute the task. These characteristics also enable the cooperation between agents, while considered as complementary resources for the system.

4.8. A recursive groups definition

The groups in SCALA are defined via the following way:
- The type of the group,
- The members (type: agent or group, and number),
- The roles composing the group,
- Group Goal: the current global goal of the group,
- The sub-groups: the groups inside the group,
- The meta group: the group which it belongs,
- The type of organisation,
- The type of cooperation

- **name**: Group2,
- **type**: Section,
- **members**: Two Aircraft,
- **roles**: 1 Leader, 1 Wingman,
- **Group Goal**: To destroy the bandit
- **Sub-groups:** none,
- **Meta group**: Group 1,
- **Organisation**: Hierarchical,
- **Cooperation**: Delegation

- **name**: Group1,
- **type**: Division,
- **members**: 2 Sections,
- **roles**: 1 escort, 1 Bomber,
- **Sub-groups:** 2 Group2,
- **Group goal:** To destroy the bandit
- **Meta group**: none,
- **Organisation**: Hierarchical,
- **Cooperation**: Delegation

fig. 3: A recursive groups definition

This notion of group enables the designer to create sophisticated mechanisms of cooperation between agents belonging to a group and even between groups. An example of teams modelling for tactical aircraft simulation can be found in [TID 98].
In our scenario, the division is composed of two sections. And recursively, the section is composed of two aircraft. The organisation is hierarchical and the cooperation is based on delegation. One can assume the leaders of the two groups take joint decision by consulting together, the organisation of Group 1 then becomes a community, but the cooperation mechanism makes essentially intervene the leaders.

4.9. The definition of the agents

In SCALA, a generic structure of agents is defined and specialised by the designer for a given application. It is decomposed in two levels:
The agent itself:
- Skills and the relevant level of specialisation,
- Resources: physical resources of the agent,
- Events: to which the agent is sensitive,
- Current goal.

The agent in its group:
- Group(s): the group(s) it belongs,
- Current role(s): the role(s) it holds in each group(s),
- Possible role(s): the other role(s) it can eventually take in each group,
- Communication Protocol: the protocols of cooperation with the other members of the group.

4.10. The social organisations and the communication protocols

The organisations define the internal relations inside the groups. Given an organisation, the agents can have different roles depending on the situation. These roles can be dynamically assigned. For the moment, we distinguish two types of organisations: hierarchical and community. The organisation can also evolve (under certain conditions) during the simulation depending of the current topology of the graph and of the predefined mechanisms (a hierarchical structure is fit to treat a task naturally decomposed into sub-tasks).
The communication protocols depend on the type of the organisations. For instance, in a hierarchical organisation, the agent which holds a manager role can delegate a task or a set of tasks to another agent or to a group via a simple point-to-point message sending. Whereas in a community, a network is built between several agents to dynamically exchange their tasks or collaborate for their execution. An example of such mechanism is the *Contract Net protocol,* which have been implemented in SCALA.

4.11. The simulation process "event to plans"

Each new relevant event that occurs during the simulation triggers a new instance of the related sub-graphs. The sub-graphs tasks are dynamically distributed to some agents according to their skills, roles and resources. The distribution process (fig. 4) depends on the organisation type of the group to which belong the sensitive agents. The sub-graphs (a multi-agent plan) are split into mono agent plans (lists of partially ordered tasks) as a network of dependencies between the activities of agents. Those dependencies are a representation of the constraints of the graph. Thus, the agents coordinate their activities while taking into account these dependencies and environment changes.

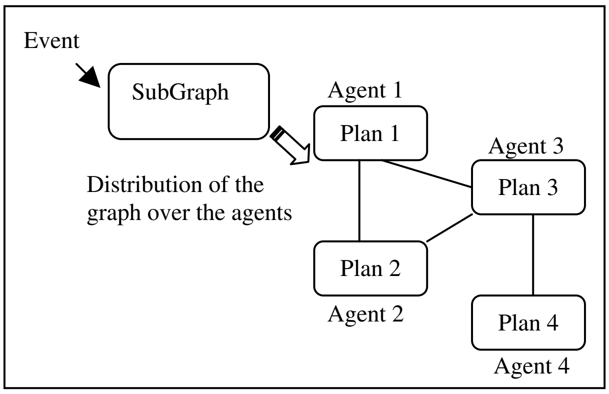


fig.4: Network of dependencies

The SCALA Grapher

Numerous applications can be modelled. To define these applications, we developed a tool to easily design the graphs of dependencies that are associated to an event name, and also build a library of reusable behaviours (the Event/goals pairs). SCALA automatically interprets them and generate the code of the simulation. Only the real contents of the tasks have to be implemented for each application, the mechanisms providing the strategies of cooperation lie on the information entered by the designer through the steps of the methodology. In SCALA, the reasoning part (including coordination between agents) is entirely independent of the basic behaviours.

The activities' scheduler

Another tool to monitor the mission is a scheduler diagram that represents the activities of the agents. On this diagram can be seen the multi-agent behaviour: the evolution of the simulation, the coordination between agents, the arrival of new events.

5. Perspectives

This first approach in SCALA encounters certain limits: the lack of temporal aspects necessary to meet the simulation requirements. Those limits could be critical factors. In addition, we are interested to explore reactive planning in order to manage the environment dynamic (arrival of new goals or events). Consequently, SCALA is extended to take into account time constraints when new events occur. It is also a critical factor when agents have to hold temporal objectives. Furthermore, it is obvious that the behaviour of a group of agent will depend on the available time they have to react.

One of the domains of application of SCALA is tactical aircraft simulation. This domain is characterised by a highly dynamic (unpredictable) and uncertain environment, that implies for the agents to plan reactively when new events occur and sometimes to reorganise themselves. But, the agents have also to respect time constraints in the execution of their tasks and so, the reorganisation and coordination of their activities have to take into account this constraints. Thus the work in progress attempts to extend the graph of dependencies by the notion of time in the tasks, and time constraints on the goals of the agents. These constraints can be assimilated to temporal objectives that are critical factors to achieve successfully the mission. Our approach is real-time execution driven.

6. Conclusion

Through the experience of the use of JACK and the development of SCALA, we have seen that the agent-oriented programming fits to the development of complex system evolving in dynamic environments. The high level of abstraction of the agent approach makes easier the modelling of complex systems, especially the modelling of distributed applications, where the entities (the agents) have to perform tasks autonomously and to interact with others. The designer can focus on sophisticated mechanisms such as coordination and cooperation between the components of the system without getting tangled up in the development of communications for messages sending, or other low-level developments such as multi-threads, etc.

Furthermore this approach allows to easily separating reasoning from actions, and so improves the reusability of the implemented behaviours. In the case of aerial missions simulations, the advantage of such programming is to make reasoning and physical models completely independent. It makes simpler the addition of new behaviours and introduces flexibility in the system. New events can easily be taken into account and managed by the system. These aspects are very interesting for designing systems submerged in a real world (in simulation or in reality), when flexibility, reactivity and adaptation are crucial (responses of new events). Interactivity with human users of the system is also facilitated.

SCALA project proposes a functional approach to design complex systems and provides a tool to rapidly setting up simulations. The designer has to model the global behaviour of the system as a graph supporting functional constraints. The connectors represented in the graph of dependencies of SCALA enable an easier coordination and organisation of the agents driven by their activities. Another important contribution in the design of complex systems is the fact that different behaviours can be easily obtained by only modifying the links and connectors in the graph of dependencies of SCALA and simulated without new compilations. New behaviours are thus specified at a high-level of modelling and then directly interpreted by SCALA to generate the code of the related simulation.

An interesting application of our work is the domain of tactical aircraft simulation, to rapidly prototype tactical behaviours of well-known aircraft or new ones.

Acknowledgements

The author would like to thank Frédéric MARC who works on the related project in the frameworks on his PhD. on Computer Science at LIPN (Laboratory of Computer Science of PARIS 13) and Dassault-Aviation.

References

[BRIOT 01] BRIOT J.-P., DEMAZEAU Y., "Principes et architectures des systèmes multi-agent", Edited by Hermes, pp. 17-70, 2001.

[HOW 01] HOWDEN N., RONNQUIST R., HODGSON A. and LUCAS A., "JACK Intelligent Agents^{TM} Summary of an Agent Infrastructure", *5th Conference on Autonomous Agent*, 2001.

[JAR 01] JARVIS J., MAISANO P, "Jack Intelligent Agents^{TM} User Guide", Release 3.1, Agent Oriented Software Pty. Ltd, Mars 2001.

[JEN 98] JENNINGS N. R.,WOOLDRIDGE M., SYCARA K., "A roadmap of agent research and development", *Int. Journal of Autonomous Agents and Multi-Agent Systems*, vol. 1, n°1, pp. 7-38, 1998.

[RAO 95] RAO A. S., GEORGEFF M. P., "Modeling rational agents within a BDI architecture", *In* J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, pp. 473-484. Morgan Kaufman Publishers, San Mateo, 1991.

[TID 98] TIDHAR G., HEINZE C., SELVESTREL M., "Flying Together: Modelling Air Mission Teams", *Applied intelligence*, vol. 8, pp. 195-218, 1998.

[WOO 95] WOOLDRIDGE M., JENNINGS N. R., "Intelligent Agents: Theory and practise", *The Knowledge Engineering Review*, 10(2):115-152, 1995.

[WOO 99] WOOLDRIDGE M., "Intelligent Agents", In Multiagent System: A Modern Approach to Distributed Artificial Intelligence, Edited by WEISS G., pp. 27-77, 1999.

This page has been deliberately left blank

Page intentionnellement blanche